Projet de Graphes, Algorithmes et Modélisation

Master 1 Informatique,
Aurélien Cavelan
Léo Rousseau

Table des matières

1 Compilation et Utilisation

1.1 Compiler le Projet

L'archive du projet est composée de plusieurs répertoires : Un répertoire **src** qui contient les sources du projet et un répertoire **doc** qui contient les sources tex du présent rapport. Un makefile à la racine du projet permet de compiler les sources.

```
1 make
```

1.2 Utiliser le Logiciel

La compilation va générer un répertoir **bin** qui va contenir l'exécutable du logiciel. Pour lancer le logiciel :

```
1 ./bin/fleury
```

Le graphe à tester devra être passé au programme sur l'entré standard selon le format fixé. Après calcul sera affiché l'ordre de parcours des nœuds du graphe dans le cycle s'il existe.

1.3 Utiliser le Générateur

Le Projet contient également un générateur de graphes complets. Pour l'utiliser, il faut le compiler :

```
1 make generator
```

Et lui passer en paramètre K le nombre de nœuds dans le graphe :

```
1 ./bin/generator K
```

Enfin, il est possible de combiner le générateur avec le lociciel afin de ne pas avoir à saisir manuellement un graphe :

```
1 ./bin/generator K | ./bin/fleury
```

2 Algorithme et Implémentation

2.1 Implémentation

2.1.1 Structure de données

Le projet s'appuie sur une structure de nœud deffinie de cette façon :

```
1 struct node_t
2 {
3     int id;
4     struct node_t ** neighbours;
5     int degree;
6 };
```

Un graphe est un tableau dynamique de nœuds. Chaque nœud contient un identifiant **id**, un ensemble de voisins **neighbours** et le nombre de voisin qu'il possède : **degree**.

Nous utilisons aussi une représentation des arêtes par couple d'entiers où chaque entier est l'identifiant d'un des nœuds.

```
1 struct edge_t {int u; int v;};
```

Cette structure est utilisée lors de la lecture sur l'entrée standard et la création d'un graphe.

Nous avons aussi défini un ensemble de fonctions permettant d'utiliser un graphe comme un objet. Ces fonctions permettent nottamment la lecture, la création, la suppression et le parcour d'un graphe. Les fonctions **graph_darken_edge** et **graph_undarken_edge** servent respectivement à marquer et démarquer une arrête lors d'un parcours. Une arête marquée n'est plus visitable.

```
1 int graph_create_from_list (struct node_t * g, int n, int m, struct
      edge_t * e, int * count);
2 int graph_read_from_std (struct node_t ** gp, int * np, int * mp);
3 void graph_destroy (struct node_t * g, int n);
4 void graph_darken_edge(struct node_t * a, struct node_t * b, int x);
5 void graph_undarken_edge(struct node_t * a, struct node_t * b);
6 void graph_dfs_display (struct node_t * g, int n);
```

2.1.2 Implémentation de l'algorithme

L'algorithme est implémenté via la fonction **fleury** dans le fichier **main.c**. Une fois que le graphe à été lu et initialisé, cette fonction va y rechercher un cycle eulerien et, s'il existe, il sera affiché (ordre de parcours des nœuds).

La fonction **fleury** s'appuie sur la fonction **pick_next**. Elle sert à trouver l'identifiant du prochain nœud à visiter étant donné un nœud de départ. Pour se faire, elle effectue un parcours en profondeur **is_not_isthme_dfs**. Ce test sert à

vérifier qu'il existe un autre chemin entre un nœud source et un nœud destination que l'arrête qui les relie directement. Si ce test est valide, alors cela signifie que l'arrête n'est pas un isthme et peut donc être visitée (noircie). Un nouveau noeud est ajouté à la solution, sinon il faut tester les autres voisins.

2.2 Étude de Complexité

Notre implémentation de l'algorithme cherche m arêtes. À chaque fois elle peut parcourir jusqu'à n voisins et pour savoir si un voisin est bon elle effectue un parcours en profondeur en $O(n+m)$; soit : $m*n*(m+n)$. Après développement on obient : m^2*n+n^2*m or, $m > n$ donc la complexité de notre implémentation est $O(m^2*n)$. On peut observer sur la courbe ci dessous la croissance du temps d'exécution de l'algorithme (en secondes) en fonction du nombre de nœuds dans un graphe (complet). On en déduit qu'il s'agit bien d'une croissance quadratique.